USER CONTROLLED ENVIRONMENT

By

ASHISH PINNINTI

B-Tech, Vellore Institute of Technology, India, 2010

A REPORT

Submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Computing and Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2015

Approved by:

Major Professor
Mitchell L. Neilsen

Copyright

Abstract

The mobile world is rapidly changing: Smartphones have gone from portable messaging and email devices to streaming-video machines that surf the Web at blazing speed. Now-a-days a smartphone can provide computing capabilities, wireless communication capabilities, run software and perform other tasks just like any traditional computer. These amazing features of a smartphone and Open Source Android market helped in the development of this project. The purpose of this project is to develop an Android application for controlling various elements of user environment.

User Controlled Environment is an Android application for home. The environment consists of smart lights, an Android mobile devices for playing music and a display. The application sends the user's preferred settings to the environment and the respective settings are applied. The preferences are displayed on the screen. The user will be able to view and adjust a variety of environmental preferences. The preferences include the light's color, light intensity, and the music. When the user exits the application the environment goes to a default state. The users can set preferences which include moods, seven colors of light, three levels of light intensity and songs that the users can select.

Table of Contents

List of Figures

List of Tables

Acknowledgements

Firstly, I would like to express my love and gratitude to my family and teachers. I would like to express my gratitude to my major professor, Dr. Mitch Neilsen, for his guidance throughout my project development. He gave me a great level of independence while I am developing this project. His knowledge in the technical details of this project helped me to a great extent. I would also like to thank my committee members, Dr. Daniel Andresen and Dr. Torben Amtoft, for serving on my committee and guiding me during my stay here. I would like to extend my sincere thanks to Professors of Computing and Information Sciences at Kansas State University for giving me the opportunity to take various courses under them. I would also like to acknowledge the academic and technical staff of the CIS department and all others who directly or indirectly supported and helped in completion my graduate study. I would also like to thank the department of Computing and Information Sciences for sponsoring my education for the past three semesters.

Chapter 1 - **Introduction**

1.1 Project Description

User Controlled Environment is home automation Android application that helps us in controlling the various elements of room environment. The various elements this environment consists of are the Phillips Hue Smart Lights, an Android mobile devices for playing music and display preferences. The application applies the user's preferred settings to the environment. The user will have the capability to view and change a variety of environmental preferences. The preferences could be the light's color, the light intensity, and the music. As the user exits the application the environment sets back to a default state. The users can set preferences which include seven colors of light, three levels of light intensity and songs that the users can select. The users can set their preferences again and the corresponding changes will be applied to the environment.

1.2 Motivation

The mobile environment provides users with a hands free home automation system. The usage of a smartphone to control the environment enables us to replace multiple remote controls with limited capabilities. In addition any open source home automation equipment can be integrated with this mobile application.

Chapter 2 - **Background**

2.1 Android OS

Android is a mobile operating system (OS) based on the Linux kernel and currently developed by Google. Android gives you a world-class platform for creating applications. Android's open nature has encouraged a large community of developers and enthusiasts to use the open-source code as a foundation for community-driven projects. There are various Getting Started, Reference Guides, developer SDK, API documentation, and design guidelines available online that will give everything one needs to start developing apps.

2.2 Android Architecture

Android operating system is a stack of software components which is roughly divided into five sections and four main layers as shown below in the architecture diagram.

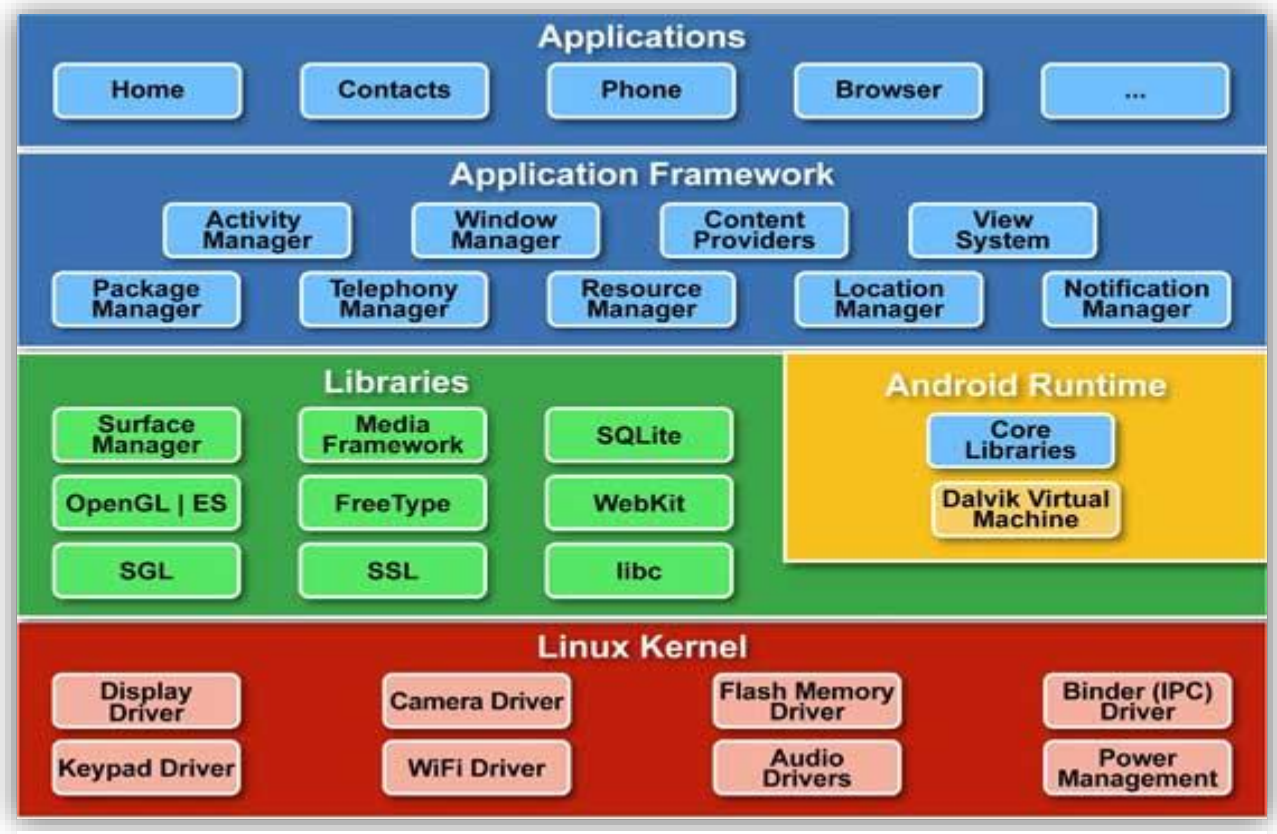

Figure 2-1 Android Architecture [10]

- Linux kernel - At the bottom of the layers is Linux - Linux 2.6 with approximately 115 patches. This provides basic system functionality like process management, memory management, device management like camera, keypad, display etc. Also, the kernel handles all the things that Linux is really good at such as networking and a vast array of device drivers, which take the pain out of interfacing to peripheral hardware.
- Libraries - On top of Linux kernel there is a set of libraries including open-source Web browser engine WebKit, well known library libc, SQLite database which is a useful repository for storage and sharing of application data, libraries to play and record audio and video, SSL libraries responsible for Internet security etc.
- Android Runtime - This is the third section of the architecture and available on the second layer from the bottom. This section provides a key component called Dalvik Virtual Machine which is a kind of Java Virtual Machine specially designed and optimized for Android. The Android runtime also provides a set of core libraries which enable Android application developers to write Android applications using standard Java programming language.
- Application Framework - The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications.
- Applications - One will find all the Android application at the top layer. One will write application to be installed on this layer only. Examples of such applications are Contacts Books, Browser, and Games etc.

2.3 Data Storage

Android provides several options for you to save persistent application data. The solution choice depends on your specific needs, such as whether the data should be private to your application or accessible to other applications and how much space your data requires.

The data storage options are the following:

- Shared Preferences - Store private primitive data in key-value pairs.
- Internal Storage - Store private data on the device memory.
- External Storage - Store public data on the shared external storage.
- SQLite Databases - Store structured data in a private database.

- Network Connection - Store data on the web with your own network server.

This application primarily uses Shared Preferences as a storage option as all the preferences can be stored as primitive data in key-value pairs.

2.4 Connectivity

Android provides rich APIs to let your app connect and interact with other devices over Bluetooth, NFC, Wi-Fi in addition to standard network connections. Most network-connected Android apps will use HTTP to send and receive data. Android includes two HTTP clients:

- HttpURLConnection
- Apache HTTP Client

Both support HTTPS, streaming uploads and downloads, configurable timeouts etc.

This application uses App Engine Rest Client for Android, which offers conveniently packaged POST and GET methods based on HttpURLConnection. The Phillips hue bridge has a powerful RESTful interface, which behaves like a simple web service.

2.5 Media Playback

The Android multimedia framework includes support for playing variety of common media types, so that you can easily integrate audio, video and images into your applications. You can play audio or video from media files stored in your application's resources (raw resources), from standalone files in the file system, or from a data stream arriving over a network connection, all using MediaPlayer APIs.

This application uses MediaPlayer classes, which provides primary API for playing sound and video in the Android framework.

2.6 AsyncTask

AsyncTask enables proper and easy use of the UI thread. This class allows to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers. AsyncTask is designed to be a helper class around Thread and Handler and does not constitute a generic threading framework. AsyncTasks should ideally be used for short operations (a few seconds at the most.)

This application uses this class to perform network operations as these operations cannot be done directly on main thread.

Chapter 3 - **Requirement Analysis**

3.1 Requirement Gathering

The project started basically by brainstorming the ideas related to the various elements in the room environment. The idea involved network and different hardware elements such as lights, Android Smartphone and a Speaker. The next step was to choose the technologies that have to be used for controlling these different elements. Then fortunately found Philips hue Lights that can be controlled using a powerful RESTful interface, which behaves like a simple web service. The other element of the environment is Music which can be controlled by Android phone itself. Android operating system comes with Media Player API for playing music. Integrating hue into music is the main purpose of this application.

3.2 Requirement Specification

3.2.1Software Requirements

Software Requirements for developing the application:

Operating Systems: Windows XP (32-bit), or Windows 8 (32- or 64-bit)

The Eclipse ADT bundle

Eclipse + ADT plugin

Android SDK Tools

Android Platform-tools

A version of the Android platform

A version of the Android system image for the emulator

Software Requirements for running the application:

1 – Android phone with version 4.4 and above

Philips hue API

Wi-Fi Internet

3.2.2Hardware Requirement

Hardware Requirements for developing the application:

Processor: Intel Core 2 Duo or higher

RAM: 2 GB

Hardware Requirements for running the application:

1 – Display with HDMI connector

1 – MHL cable or Belkin Miracast

1 – Phillips Hue Smart Lights with version 8.5W, 600 lm

1 – Phillips Hue Bridge

1 - Local Wi-Fi Router

1 - LAN wire to connect the router and the hue bridge

Chapter 4 - **System Architecture and Design**

4.1 System Architecture

The following is the system architecture of the User Controlled Environment application.

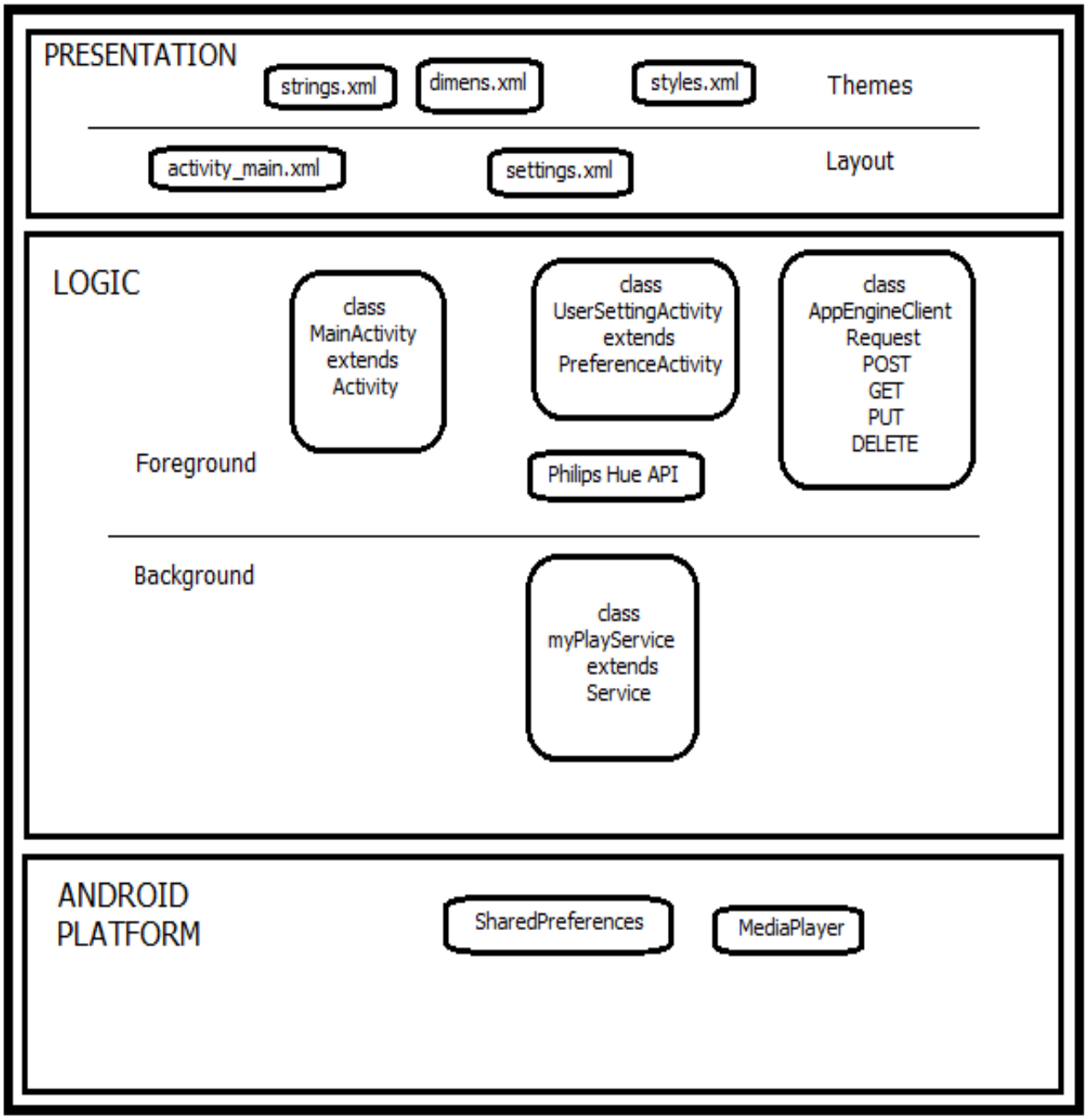

Figure 4-1 System Architecture

The user can either choose different moods available on the main screen. The preferences specific to the selected mood is sent to Phillips Hue Bridge using the App Engine REST Client.

The bridge controls the smart lights using the Phillips Hue API. Also the information about the music is sent to myPlayService using an Explicit Intent. The user can also change the preferences using the options menu and preferences activity. This is accomplished using UserSettingActivity class and settings xml. The chosen preferences are stored in SharedPreferences, read from it and again sent to Phillips Hue Bridge using the App Engine REST Client. The myPlayService uses Android's MediaPlayer to play the selected track in the background.

Lights are connected to the bridge via an open standards protocol called ZigBee Light Link [1]. Using this protocol we can directly talk to the bulbs. The protocol uses AES 128 encryption to protect lighting network against unauthorized use. The following diagram depicts the communication between mobile device, Phillips Hue Bridge and smart bulb.

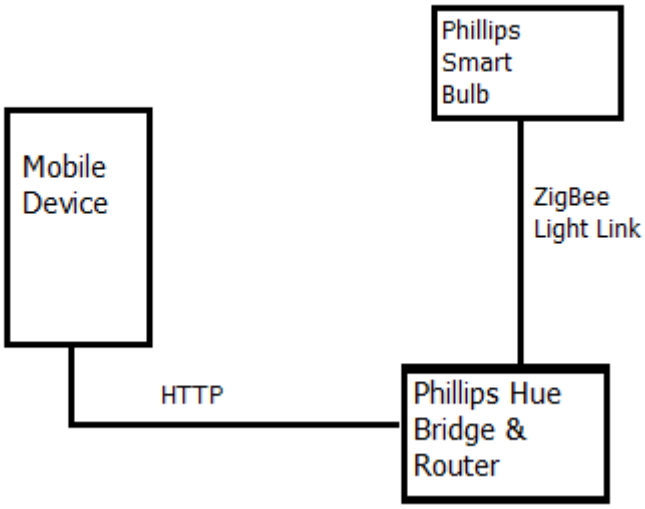

Figure 4-2 ZigBee Light Link Communication

4.2 System Design

The Unified Modeling Language (UML) and Class diagrams helped in depicting the various aspects of the overall application design. They show different functionalities, user interactions, classes and their relationships. They helped in analysis, design, and implementation of the application.

4.2.1 Use case diagram

The following Unified Modeling Language (UML) diagram offers a way to visualize a app's architectural blueprint, including elements such as: different functionalities, individual activities, how they can interact with other activities, how the app will run, how user interact with app, external user interface etc.

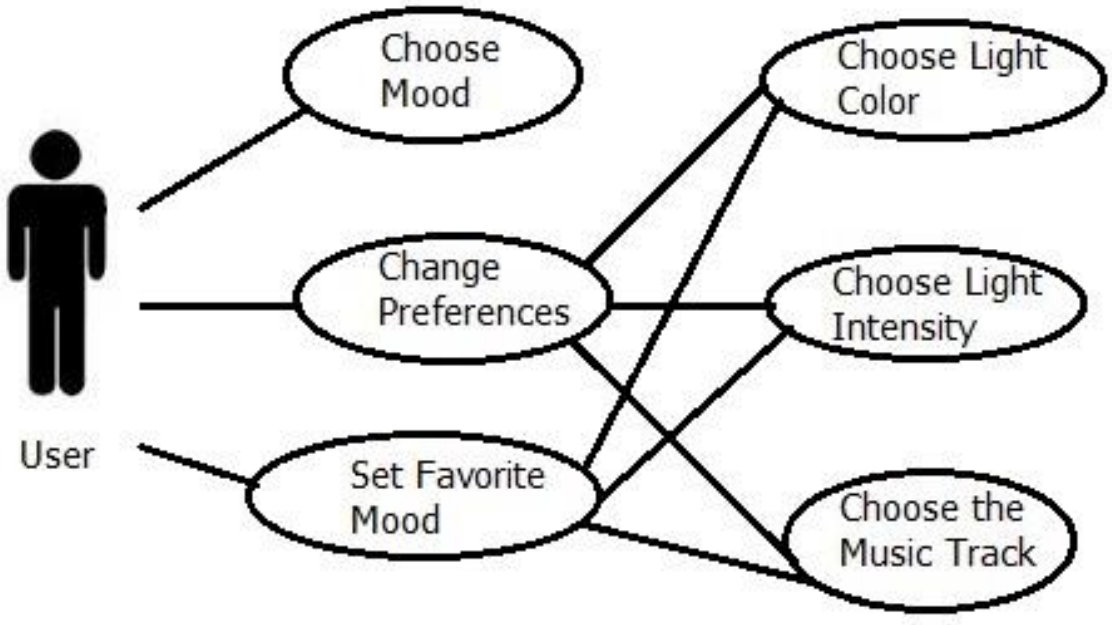

Figure 4-3 Use Case Diagram

4.2.2 Class Diagram

The class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of the application by showing the system's classes, their attributes, operations (or methods), and the relationships among objects. The class diagram is the main building block of object oriented modelling. It is used both for general conceptual modelling of the systematics of the application, and for detailed modelling translating the models into programming code.

The following class diagrams shows different classes that represent different screens or activities specific to the application. These also captures the interaction with the REST client.

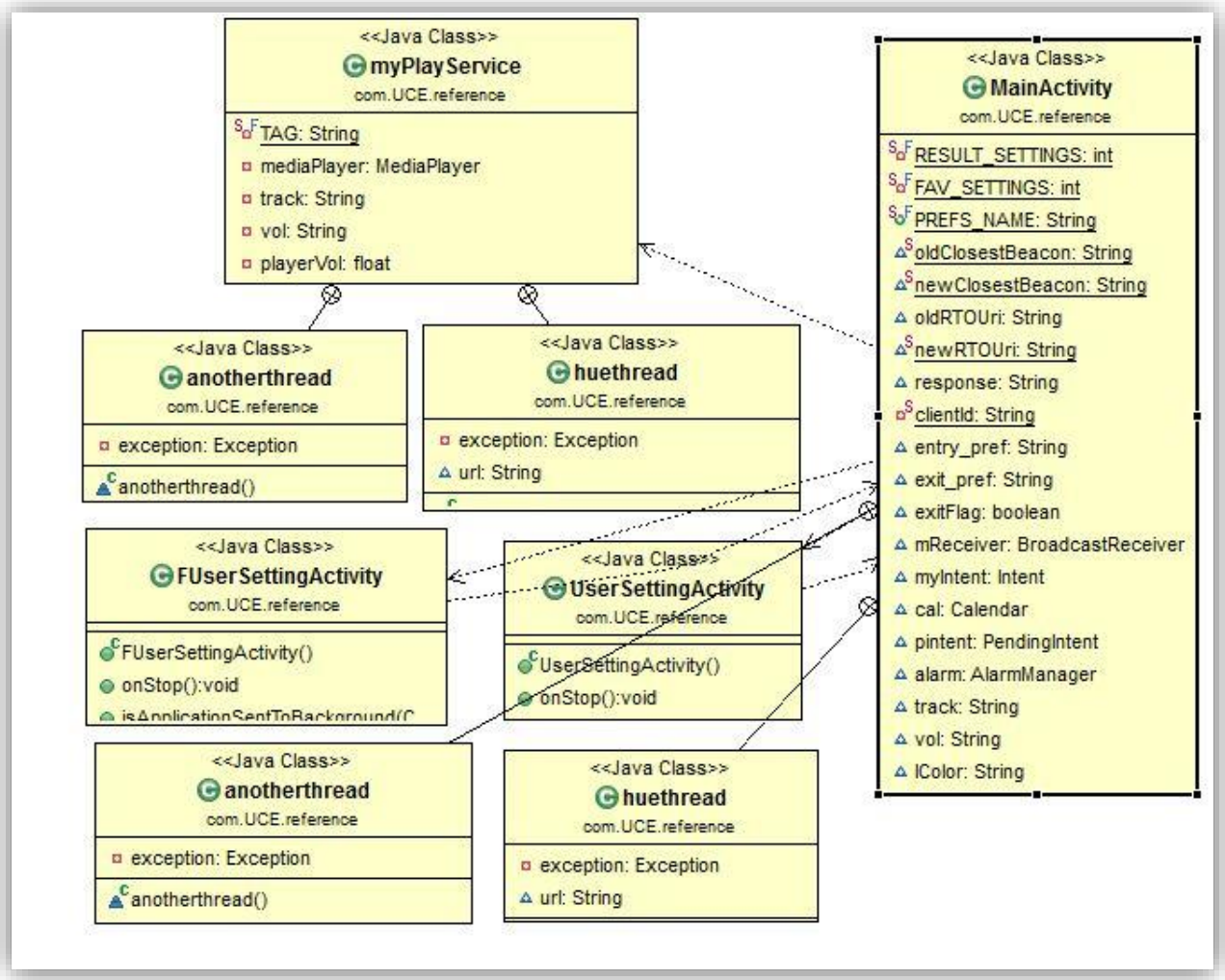

Figure 4-4 Class Diagram 1

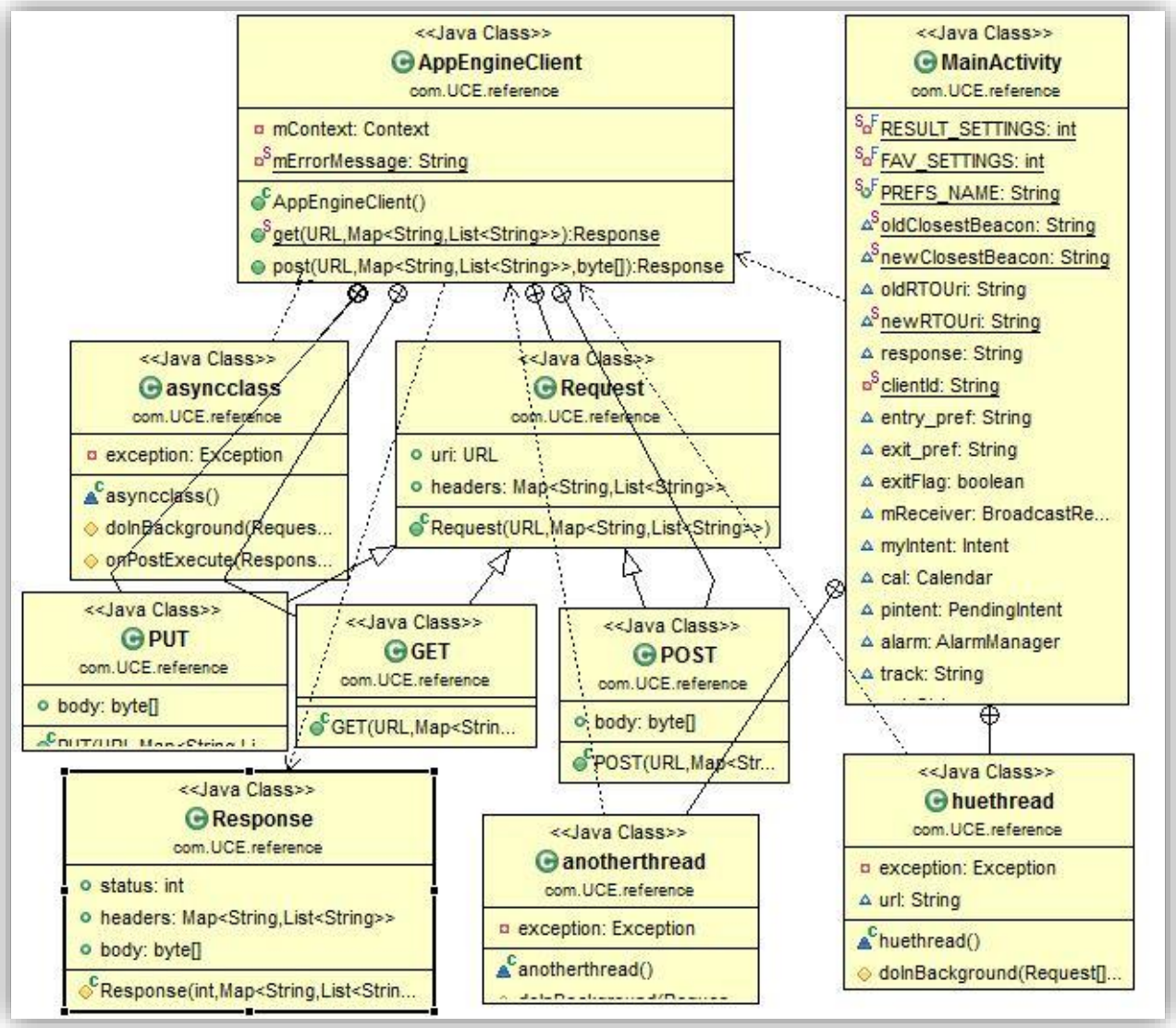

Figure 4-5 Class Diagram 2

Chapter 5 - **Android Framework Components**

Android apps are written in the Java programming language. The Eclipse ADT Bundle provides everything you need to start developing apps, including the Android SDK tools and a version of the Eclipse IDE with built-in ADT (Android Developer Tools) to streamline your Android app development. May need to download different versions of SDK tools and platforms using the SDK Manager. The Android SDK tools compile your code—along with any data and resource files—into an APK: an Android package, which is an archive file with an .apk suffix. One APK file contains all the contents of an Android app and is the file that Android-powered devices use to install the app.

5.1 App Manifest

Every application must have an AndroidManifest.xml [2] file in its root directory. The manifest file presents essential information about your app to the Android system, information the system must have before it can run any of the app's code. Among other things, the manifest does the following:

- It names the Java package for the application. The package name serves as a unique identifier for the application.
- It describes the components of the application — the activities, services, broadcast receivers, and content providers that the application is composed of. It names the classes that implement each of the components and publishes their capabilities. These declarations let the Android system know what the components are and under what conditions they can be launched.
- It determines which processes will host application components.
- It declares which permissions the application must have in order to access protected parts of the API and interact with other applications.
- It also declares the permissions that others are required to have in order to interact with the application's components.
- It declares the minimum level of the Android API that the application requires.
- It lists the libraries that the application must be linked against.

The following is the AndroidManifest.xml file present in this application:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.UAE.ibeaconreference"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="18"
        android:targetSdkVersion="19" />

    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"
/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"
/>
    <uses-permission android:name="android.permission.GET_TASKS" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"
/>

    <application
        android:allowBackup="true"
        android:icon="@drawable/uceapp"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.UCE.reference.MainActivity"
            android:label="@string/app_name"
            android:permission="android.permission.INTERNET"
            android:screenOrientation="portrait" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name="com.UCE.reference.UserSettingActivity"
            android:label="@string/app_name"
            android:screenOrientation="portrait"
            android:theme="@style/PreferencesTheme" >
            <intent-filter>
                <action android:name="android.intent.action.SETTING" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name="com.UCE.reference.FUserSettingActivity"
            android:label="@string/app_name"
            android:screenOrientation="portrait"
            android:theme="@style/PreferencesTheme" >
```

```
            <intent-filter>
                <action android:name="android.intent.action.SETTING" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name="com.UCE.reference.BackgroundActivity"
            android:label="@string/app_name"
            android:screenOrientation="portrait" >
            <intent-filter>
            </intent-filter>
        </activity>

        <service
            android:name="com.UCE.reference.myPlayService"
            android:enabled="true" />

        <receiver android:name="ReceivingService" >
        </receiver>
    </application>

</manifest>
```

The AndroidManifest.xml specific to this application includes permissions, different activities, and services required to run the application. The minimum SDK required to run the application is 18, while the target SDK is 19. The app icon and name is defined here in the manifest. ACCESS_NETWORK_STATE and ACCESS_WIFI_STATE permissions is required to access information such as Phillips Hue Bridge IP address. INTERNET permission is required to place REST calls etc. BLUETOOTH permission allows applications to connect to paired bluetooth devices such as external speaker. Different activities have different capabilities, the intent filters are used to define these capabilities. The MAIN and SETTING filters define the main launching and settings activities respectively.

5.2 Activities

An Activity is an application component that provides a screen with which users can interact in order to do something, such as dial the phone, take a photo, send an email, or view a map. Each activity is given a window in which to draw its user interface. The window typically fills the screen. An application usually consists of multiple activities that are loosely bound to each other.

Managing the lifecycle of your activities by implementing callback methods is crucial to developing a strong and flexible application. When an activity transitions into and out of the

different states, it is notified through various callback methods. All of the callback methods are hooks that you can override to do appropriate work when the state of your activity changes. The following skeleton activity [3] includes each of the fundamental lifecycle methods:

```
public class ExampleActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // The activity is being created.
    }
    @Override
    protected void onStart() {
        super.onStart();
        // The activity is about to become visible.
    }
    @Override
    protected void onResume() {
        super.onResume();
        // The activity has become visible (it is now "resumed").
    }
    @Override
    protected void onPause() {
        super.onPause();
        // Another activity is taking focus (this activity is about to be
"paused").
    }
    @Override
    protected void onStop() {
        super.onStop();
        // The activity is no longer visible (it is now "stopped")
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        // The activity is about to be destroyed.
```

```
        }
}
```

The following figure illustrates these loops and the paths an activity might take between states. The rectangles represent the callback methods you can implement to perform operations when the activity transitions between states.

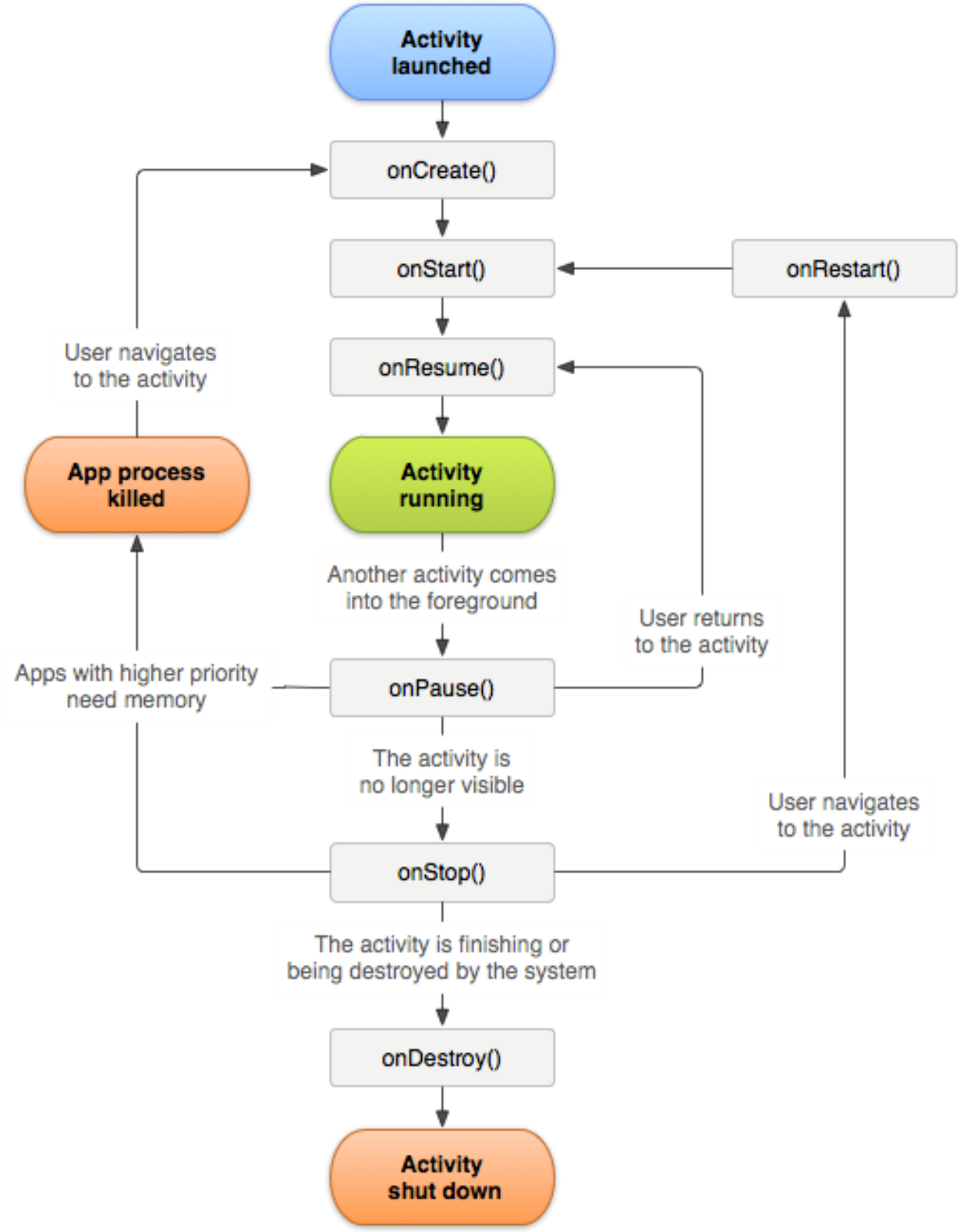

Figure 5-1 The Activity Lifecycle [4]

5.3 Intents

An Intent is a messaging object you can use to request an action from another app component. Although intents facilitate communication between components in several ways, there are three fundamental use-cases:

- To start an activity
- To start a service
- To deliver a broadcast

5.4 Services

A service is a component that runs in the background to perform long-running operations or to perform work for remote processes. A service does not provide a user interface. For example, a service might play music in the background while the user is in a different app, or it might fetch data over the network without blocking user interaction with an activity. Another component, such as an activity, can start the service and let it run or bind to it in order to interact with it. A service is implemented as a subclass of Service.

5.5 Content providers

A content provider manages a shared set of app data. One can store the data in the file system, an SQLite database, on the web, or any other persistent storage location your app can access. As such, any app with the proper permissions can query part of the content provider to read and write information about a particular person. Content providers are also useful for reading and writing data that is private to your app and not shared. A content provider is implemented as a subclass of ContentProvider and must implement a standard set of APIs that enable other apps to perform transactions. For more information, see the Content Providers developer guide.

5.6 Broadcast receivers

A broadcast receiver is a component that responds to system-wide broadcast announcements. Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. Apps can also initiate broadcasts—for example, to let other apps know that some data has been downloaded to the device and is available for them to use. Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs. More commonly, though, a

broadcast receiver is just a "gateway" to other components and is intended to do a very minimal amount of work. For instance, it might initiate a service to perform some work based on the event. A broadcast receiver is implemented as a subclass of BroadcastReceiver and each broadcast is delivered as an Intent object.

Chapter 6 - Implementation

The User Controlled Environment application is developed to help an individual in controlling the environment with various elements such as smart lights and music. The main objective of the application is to provide the user with few sample themes or moods. The different settings associated with specific mood will be applied once the user chooses the mood. The user can also apply the preferences by choosing each of the preference, the song and the light color, the light intensity level etc., independent of any mood.

The main features of the application are:

Change Preferences - Once the user opts to change the preferences, he will be taken to preference setting screen, where he will be provided with different options to choose from. The user could choose the color of the light, the brightness (Soft, Pleasant, and Bright) of the light, the song to play in the background.

Choose Mood - The application will come along with few pre-installed moods. Once the user chooses any mood, the settings specific to that mood will be applied.

The User Controlled Environment Android application is developed on Android SDK 4.4 using The Eclipse ADT bundle and Java 1.7. The user interface is developed using XML and the business logic is written in Java. The application uses SharedPreferences class that provides a general framework that allows you to save and retrieve persistent key-value pairs of primitive data types, Dalvik Debug Monitor Server (DDMS) in eclipse for debugging, LogCat to print the stack traces when necessary and platform tools such as Android Debug Bridge (adb) to lets you communicate with an emulator instance or connected Android-powered device. The total lines of code written in this application are 2611 which includes Java and XML files. Table below lists the breakdown of lines of code with respect to the language used.

Language	Number of LOC
Java	2147
Xml	464
Total	2611

Table 6-1 Lines of Code

6.1 Activity Screens

Keeping in mind that the application's user interface is everything that the user can see and interact with. The application uses Android's variety of pre-build UI components such as structured layout objects and UI controls. Android also provides other UI modules for special interfaces such as dialogs, notifications, and menus. All user interface elements in this application are built using View and ViewGroup objects. A View is an object that draws something on the screen that the user can interact with. A ViewGroup is an object that holds other View and ViewGroup objects in order to define the layout of the interface.

This application has UI elements declared in XML. Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts. We can instantiate layout elements at runtime and the application can create View and ViewGroup objects and manipulate their properties programmatically. The Android framework gives you the flexibility to use either or both of these methods for declaring and managing your application's UI. For example, one could declare your application's default layouts in XML, including the screen elements that will appear in them and their properties. One could then add code in your application that would modify the state of the screen objects, including those declared in XML, at run time. The ADT Plugin for Eclipse offers a layout preview of the XML — with the XML file opened, select the Layout tab for graphical view. This has most of the UI elements declared in XML as it enables to better separate the presentation of the application from the code that controls its behavior (separation of concerns).

The Dalvik Debug Monitor Server (DDMS) can capture screenshots from the emulator or actual device. Select Device > Screen capture.

6.1.1 Home Page

The home page displays the following:

- The current settings of the environment.

- A menu button to change preferences.
- Different icons representing in-built moods.
- Favorite mood icon.

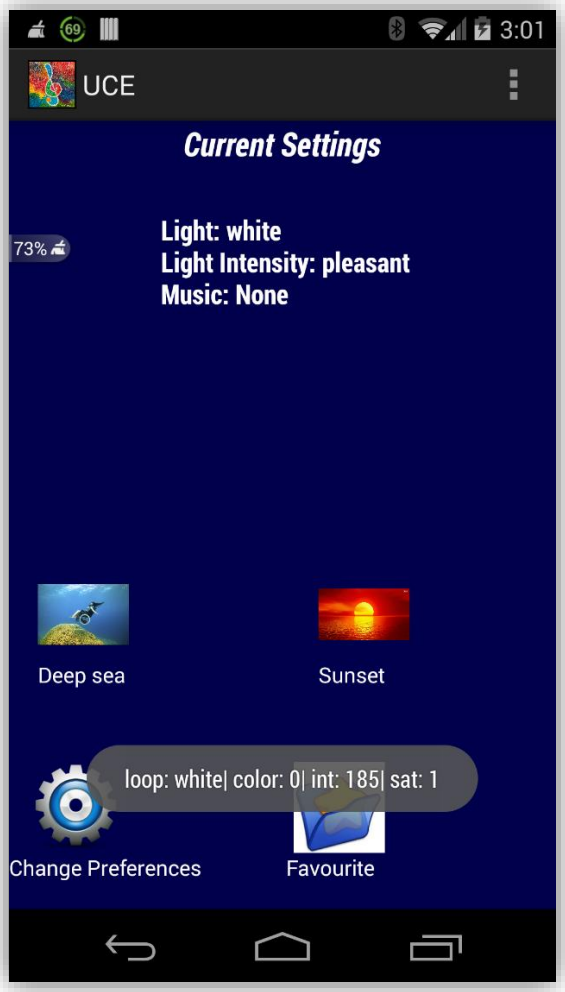

Figure 6-1 Home Screen

All the UI elements here in the home screen are declared in activity_main.xml file. This screen uses the RelativeLayout view group that displays child views in relative positions. The RelativeLayout is chosen because it can eliminate nested view groups and keep your layout hierarchy flat, which improves performance. The RelativeLayout has several TextViews and ImageViews as its child views. The header text is a TextView with static text defined in XML. The user settings text view is populated programmatically in the corresponding activity class.

showUserSettings() method populates the user settings text view using getDefaultSharedPreferences() of PreferenceManager class. If the user chooses any mood from the in-built moods, the corresponding onClick method of the image view is called. Then the corresponding settings will be applied, while the user settings text view is populated accordingly intern using showUserSettings(). The positions of these child views on the screen are adjusted using Graphical layout tab of xml file. There will be other static text views for describing the mood names. The other properties such as background color, layout width, text style, source of the image corresponding to different child views are defined in the XML. The user can also set the preferences for his favorite mood using the preferences screen and use it later as the settings will be persisted.

6.1.2 Preference Screen

The Preference screen displays the following:

- The light color.
- The light intensity.
- The song to play.

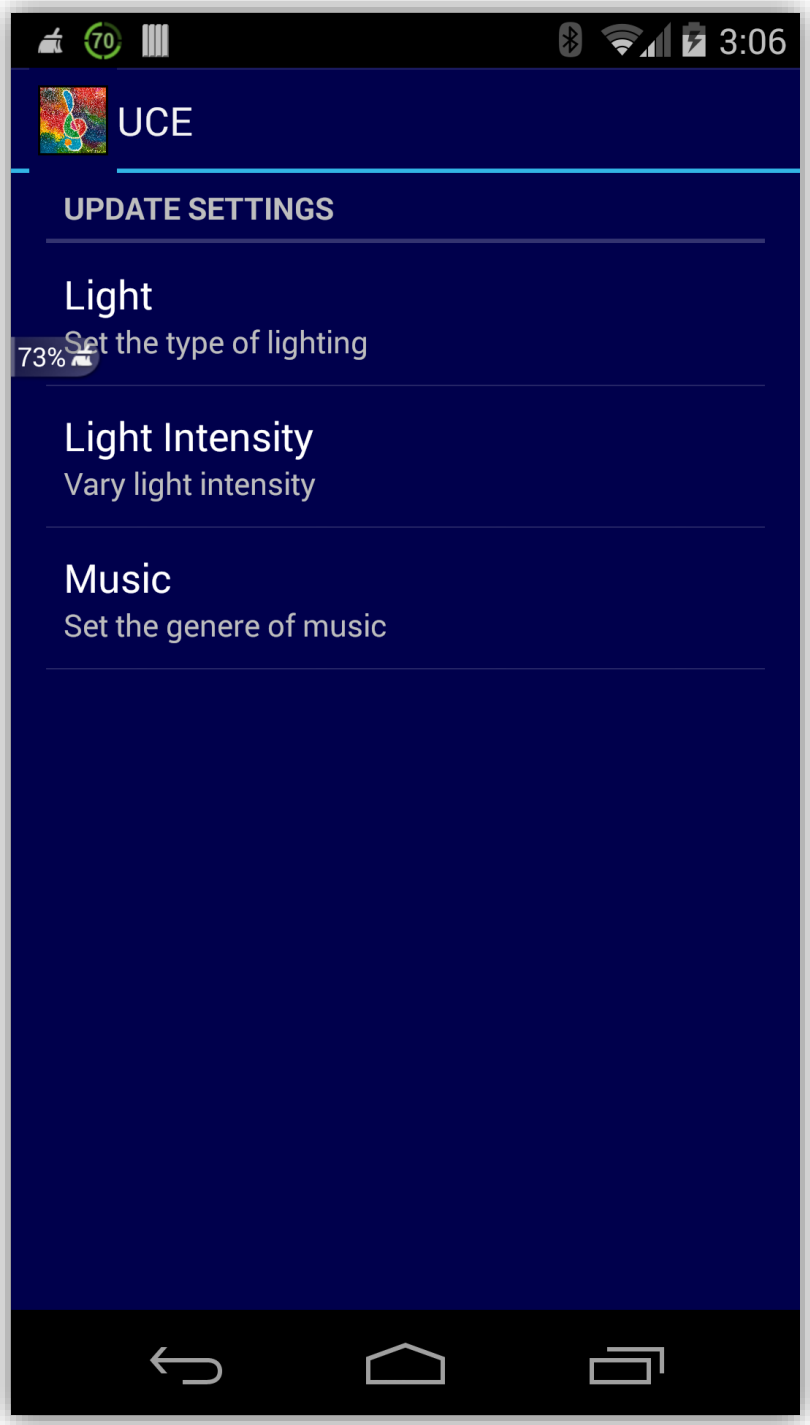

Figure 6-2 Preference Screen

When the user clicks on change preferences image view or chooses preferences from menu option, the preferences screen is displayed. This application uses startActivityForResult() of an explicit intent to launch a UserSettingActivity which extends PreferenceActivity. Here onCreate()

is implemented and the UI is defined by giving the Xml file to addPreferencesFromResource() as an argument. All the UI elements in the preferences screen is declared in settings.xml. Here the user will be able to choose different preferences such as color, intensity, music etc. The different options or values for each of these preferences are listed as resource items in other Xml file. All the items corresponding to each of these preferences are declared in arrays.xml. After the user has chosen these preferences and presses back button, the chosen settings will be saved in SharedPreferences and onActivityResult() method of the calling activity is called.

Now the saved settings are displayed on the home screen and the data is passed to huecontrol() which will place REST based network calls to hue bridge to control lights. The hue bridge has a powerful RESTful interface [5]. which behaves like a simple web service. The application gets the IP address of the bridge by using this API. Then by using this IP address we can pass the settings of the lights by reading them from Shared Preferences. The REST based network calls cannot be placed directly on the main UI thread. So AsyncTask is used to place these network calls. This class enables proper and easy use of the UI thread and performs background operations. All the REST based API calls such as PUT, POST etc. are defined in AppEngineClient class. For the music we will use service class, where different life cycle methods such as onStartCommand(). Intent object is used to pass information to service from main class. We use MediaPlayer class and pass the track information to create(). Then we use start() of the MediaPlayer class to play the music. We use Service in the application as we need to perform long-running operations, here to play music, in the background.

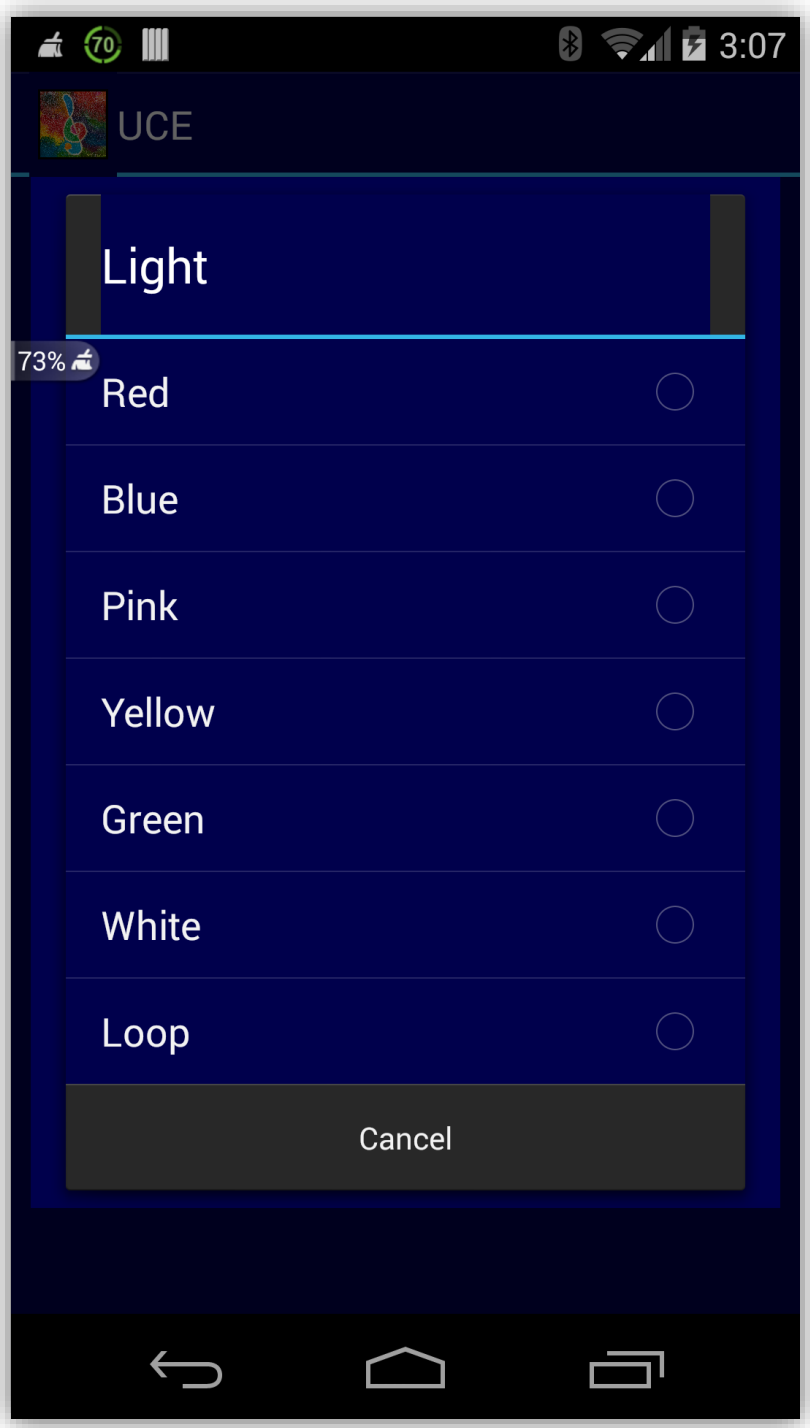

Figure 6-3 Light Color Setting Screen

When the user clicks on 'Light' setting on preference screen, the list of colors is displayed. This list of colors is stored as different resource item values in arrays.xml file. When the user chooses any color the preference screen is displayed back, so that, user can choose any additional settings.

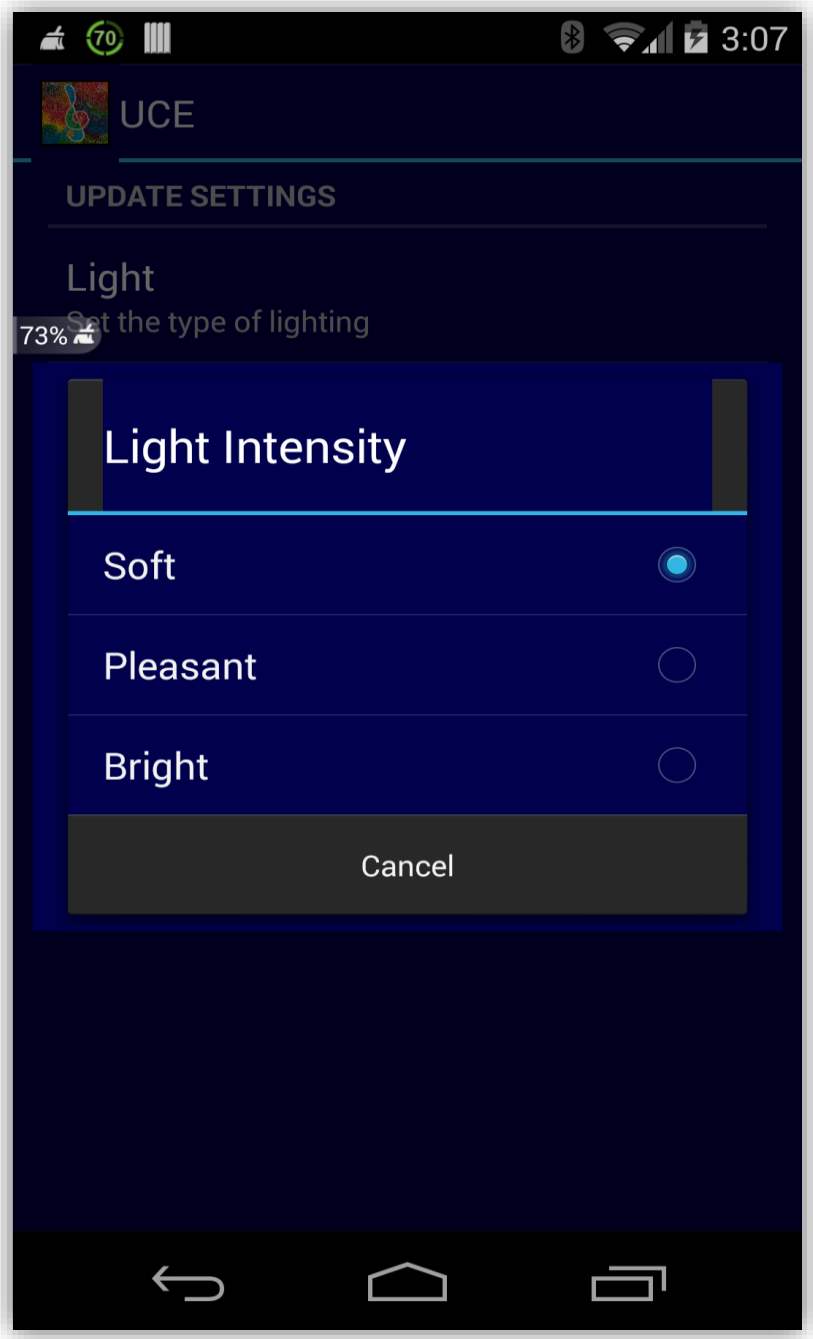

Figure 6-4 Light Intensity Setting Screen

When the user clicks on 'Light Intensity' setting on preference screen, the levels of intensity are displayed. These levels are stored as different resource item values in arrays.xml file. When the user chooses any level the preference screen is displayed back, so that, user can choose any additional settings.

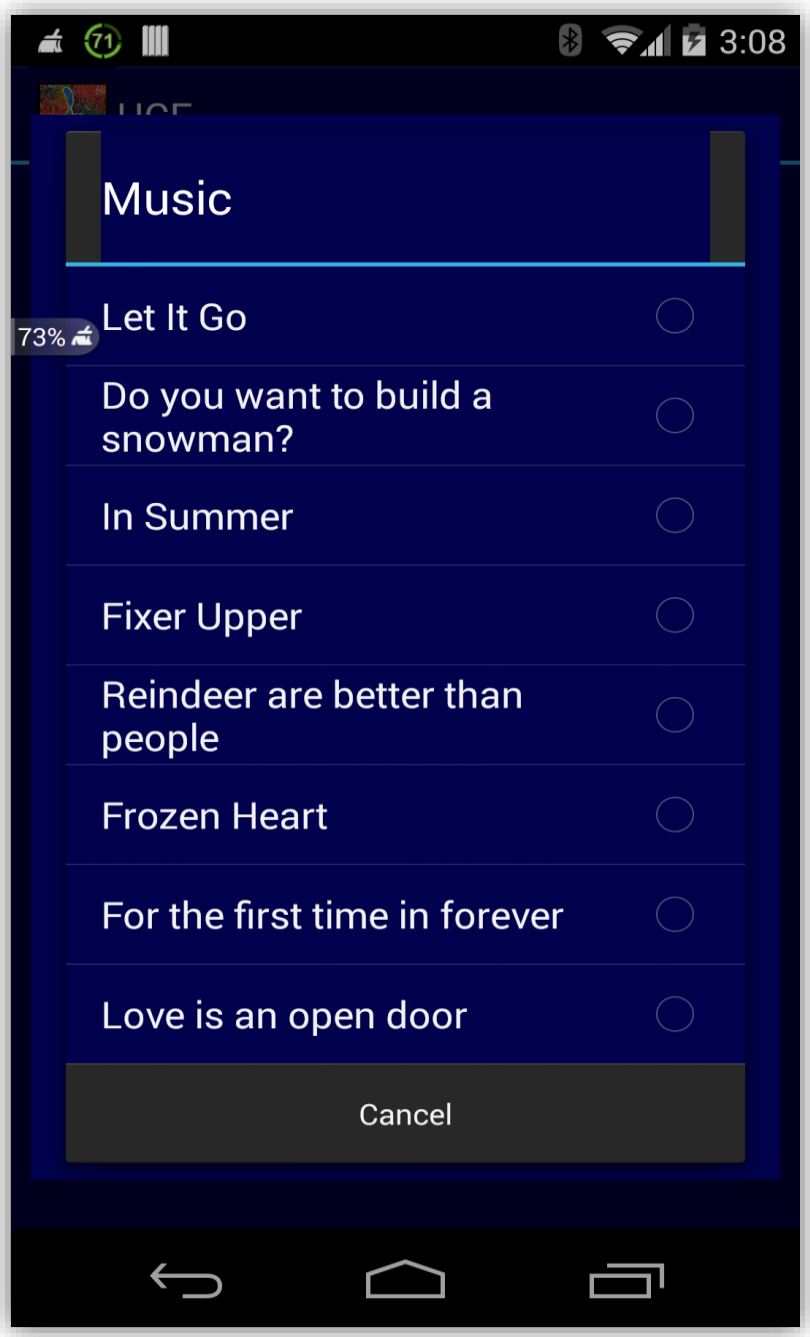

Figure 6-5 Music Setting Screen

When the user clicks on 'Music' setting on preference screen, the list of songs is displayed. This list is stored as different resource item values in arrays.xml file. The raw files of songs are stored in 'raw' folder of the app. When the user chooses any song the preference screen is displayed back, so that, user can choose any additional settings.

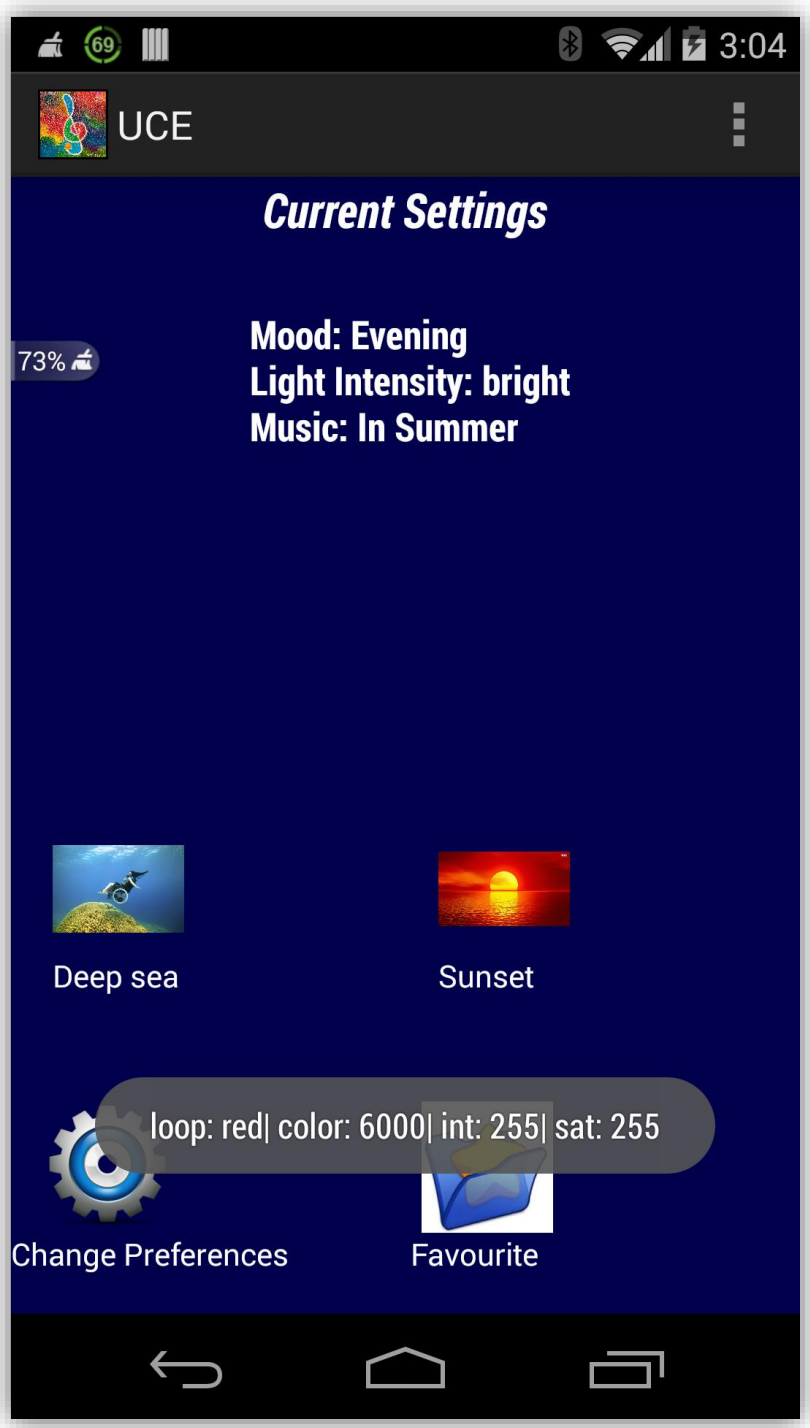

Figure 6-6 Sunset Mood Screen

When the user clicks on change sunset image view, the text views on the home screen are updated according to the settings specific to this in-built mood. At the same time, the settings are applied to the environment.

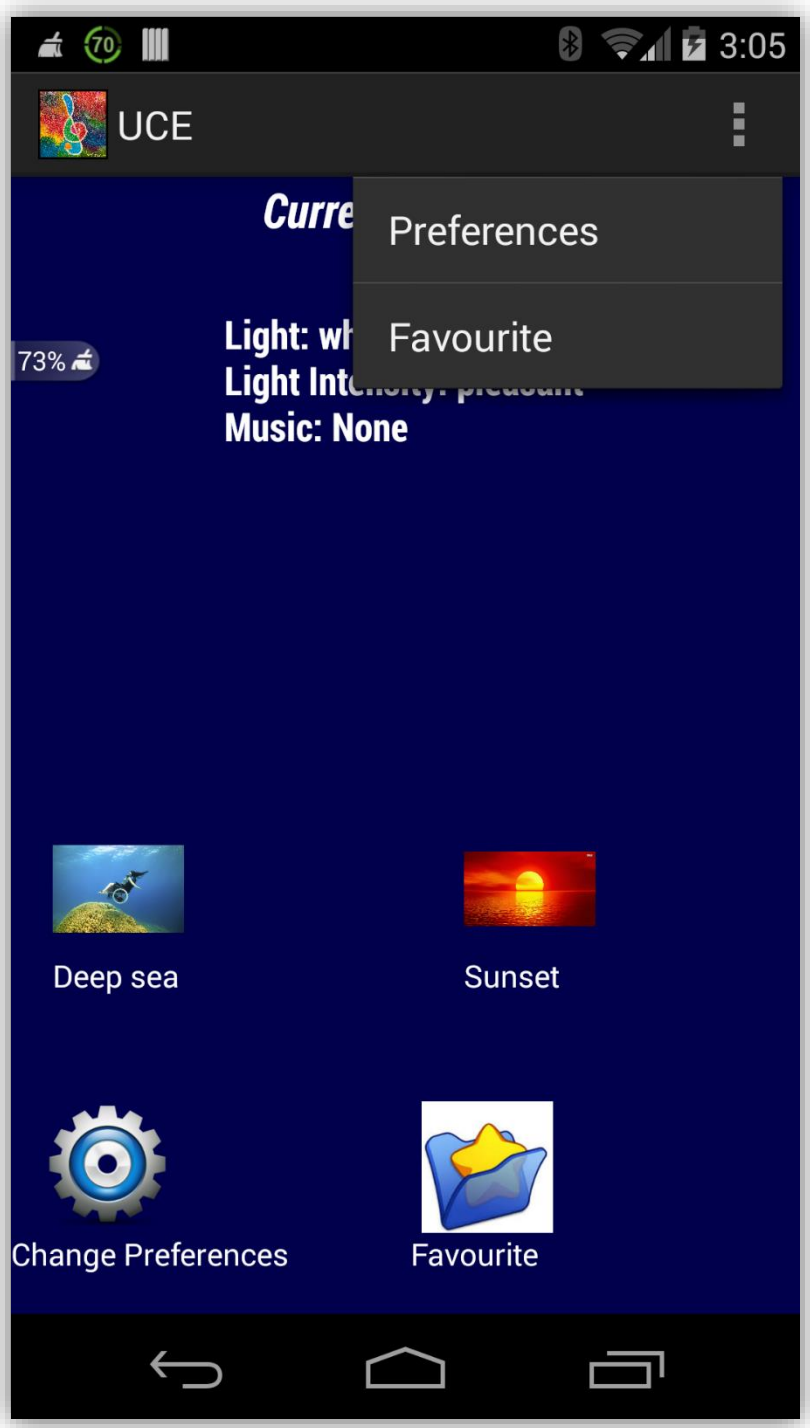

Figure 6-7 Menu Options Screen

When the user clicks on 'menu' option, the menu items 'Preferences' and 'Favorite' are displayed. The menu items are stored as item list in settings.xml file in menu folder of the app.

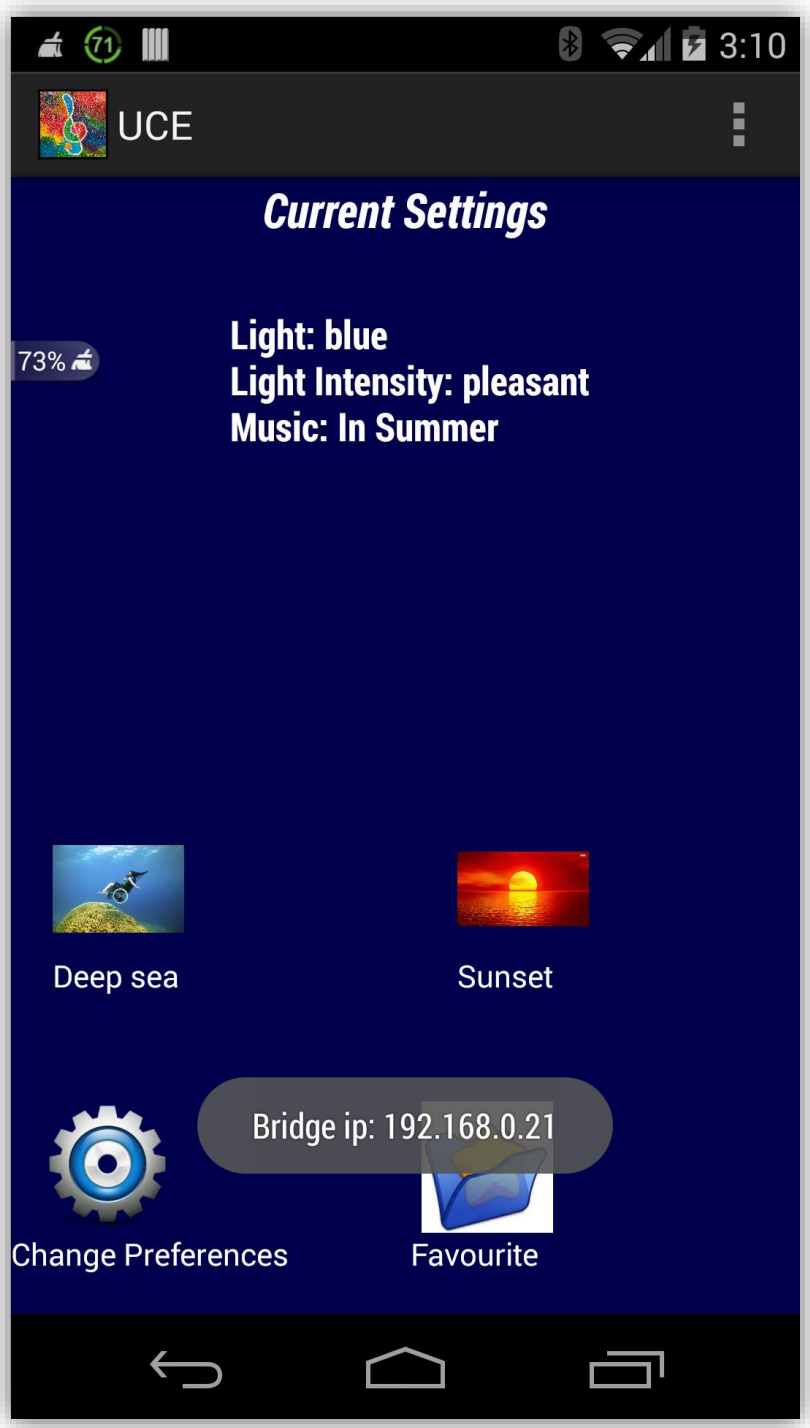

Figure 6-8 Favorite Mood Setting Screen

When the user clicks on change Favorite image view, the text views on the home screen are updated according to the settings specific to this user specific mood. At the same time, the settings are applied to the environment.

Chapter 7 - Testing

The Mobile Application testing helps to improve the quality, reliability, correctness and performance of Mobile Application. The Android SDK provides most of the tools [6] that are needed to debug applications. Dalvik Debug Monitor Server (DDMS) is a graphical program that communicates with devices. LogCat is integrated into DDMS, and outputs the messages that can be printed out using the Log class along with other system messages such as stack traces when exceptions are thrown.

7.1 Unit Testing

Unit testing is a software testing method by which individual units of the application are tested separately. In this application different screens and the related functionalities are tested. The testing is done on Nexus 5 which runs Android 4.4.

Sr. no	Test Case	Expected Result	Pass/Fail
1	On load of Home screen	Default preferences are displayed on the Home screen and are applied to the environment. The menu button to change preferences, different icons representing in-built moods and favorite mood icon are displayed.	Pass
2	On Click of any in-built mood icon.	The preferences specific to the selected mood are displayed on the Home screen and are applied to the environment.	Pass
3	On click of favorite mood icon.	The preferences set for the favorite mood are displayed on Home screen and are applied to the environment.	Pass
4	On click of menu button icon.	The menu options ‘Preferences’ and ‘Favorite’ are displayed.	Pass
5	On click of ‘Preferences’ menu option.	The ‘Preference Screen’ to set the current preferences such as The light	Pass

		color, The light intensity, The song to play etc. is displayed.	
6	On click of 'Favorite' menu option.	The 'Preference Screen' to set the Favorite mood preferences such as The light color, The light intensity, The song to play etc. is displayed.	Pass
7	On click of 'Light Color' setting on Preference Screen.	The list of colors should be displayed.	Pass
8	On click of 'Light Intensity' setting on Preference Screen.	The options of intensity levels viz. Soft, Pleasant and Bright are displayed.	Pass
9	On click of 'Music' setting on Preference Screen.	The list of songs that will come with app are displayed.	Pass
10	On selecting individual setting such as light color and on clicking 'cancel'.	The 'Preference Screen' to set the other settings such as The light intensity, The song to play etc. is displayed back.	Pass
11	On pressing the back button after setting different preferences on 'Preference Screen'.	The currently selected settings are saved, displayed on Home screen and are applied to the environment.	Pass
12	On click of 'Change Preferences' button on Home screen.	The 'Preference Screen' to set the current preferences such as The light color, The light intensity, The song to play etc. is displayed directly.	Pass

Table 7-1 Unit Test Cases

7.2 Compatibility Testing

The User Controlled Environment application is installed on different android devices such as Nexus 5, LG G3 and HTC One M8. As there are no background images and the UI is developed using Xml and Relative layouts, the app will adapt to various screen sizes.

7.3 Performance Testing

Traceview [7] helps to profile the application and find bottlenecks. It shows execution of various calls through the entire stack. The following table shows the performance with respect to different functions in the application.

Function	Time Taken (sec)
Apply Sunset Mood	0.312
Apply Current Preferences	0.328
Apply Favorite Settings	0.349

Table 7-2 Functions and Times Taken

7.4 Usability Testing

Few of my colleagues ran User Controlled Environment application on their phones and tested different functionalities of the application. They suggested few exception handling cases. All the suggested changes are implemented in the final version of the application. User should make sure that the Bluetooth on before the application is started. If at all the Bluetooth is set to off then, the application will prompt the user to set it on and terminates. It is required for the mobile to be on the same network to which the Phillips Hue bridge is connected. Sometimes if the router is not connected to internet, then mobile will use 4GLTE, but this should not happen. In order to avoid this we have to set the Mobile data to off.

Chapter 8 – Conclusion

Android is a mobile operating system (OS) that is widely used on many smart phones. Android gives a world-class platform for creating applications. Android's open nature has encouraged me to develop this project. There are various Getting Started, Reference Guides, developer SDK, API documentation, and design guidelines available online that will give everything one needs to start developing apps.

The project started basically by brainstorming the ideas related to the various elements in the room environment. The idea involved network and different hardware elements such as lights, Android Smartphone and a Speaker. Then the job is to choose the technologies that have to be used for controlling these different elements. One of the element is Philips hue Lights that can be controlled using a powerful RESTful interface, which behaves like a simple web service. The other element of the environment is Music which can be controlled by Android phone itself. Android operating system comes with Media Player API for playing music. Finally, integrated hue into music, which is the main purpose of this application.

The User Controlled Environment application is developed from scratch by following agile methodology to help an individual in controlling the environment with various elements such as smart lights and music. Once the user opts to change the preferences, he will be taken to preference setting screen, where he will be provided with different options to choose from. The user could choose the color of the light, the brightness (soft, Pleasant, Bright) of the light, the song to play in the background. The application will come along with few pre-installed moods. Once the user chooses any mood, the settings specific to that mood will be applied. This project helped me in increasing my skill set in Android Development. Learned many mechanisms such as Broadcasts, Services, AsyncTask etc. which are very specific to Android. Learned how to integrate third party APIs with any android app. Learned best practices in creating UI for android apps and how to use debugging tools such as Dalvik Debug Monitor Server (DDMS), logcat and a command line tool Android Debug Bridge (adb).

Chapter 9 - Future Work

The functionalities of User Controlled Environment application could be increased by integrating it with Nest Thermostat, which helps in controlling the temperature of the room. This is possible by using Nest's official android based client libraries or Nest's REST streaming implementation [8]. In addition, I would like to work on security evaluation of Phillips Hue API [9].

Chapter 10 - Bibliography

[1] "ZigBee Light Link Protocol," [Online]. Available: http://zigbee.org/zigbee-for-developers/applicationstandards/zigbee-light-link/. [Accessed 2014].

[2] "App Manifest," 28 October 2014. [Online]. Available: http://developer.android.com/guide/topics/manifest/manifest-intro.html.

[3] "Activity Lifecycle Callbacks," 28 October 2014. [Online]. Available: http://developer.android.com/guide/components/activities.html#ImplementingLifecycleCallbacks.

[4] "Activity States," 28 October 2014. [Online]. Available: http://developer.android.com/guide/components/activities.html#Lifecycle.

[5] "Philips-Hue-API," 21 October 2014. [Online]. Available: http://www.developers.meethue.com/philips-hue-api.

[6] "Debugging," 7 November 2014. [Online]. Available: http://developer.android.com/tools/debugging/index.html.

[7] "Traceview," [Online]. Available: https://www.youtube.com/watch?v=AJVolvNwHL8. [Accessed 9 November 2014].

[8] "Nest Thermostat API," 21 October 2014. [Online]. Available: https://developer.nest.com/.

[9] "Hacking Lightbulbs," [Online]. Available: http://www.dhanjani.com/docs/Hacking%20Lighbulbs%20Hue%20Dhanjani%202013.pdf. [Accessed 4 December 2014].

[10] "Android Architecture," [Online]. Available: http://www.tutorialspoint.com/android/android_architecture.htm. [Accessed 28 October 2014].